Construction of d-dimensional hyperoctrees on a hypercube multiprocessor

Franke Dehne, Andreas Fabri, Mostafa Nassar, Andrew Rau-Chaplin, Rada Valiveti

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Rapports de Recherche

N°1752

Programme 4

Robotique, Image et Vision

CONSTRUCTION OF D-DIMENSIONAL HYPEROCTREES ON A HYPERCUBE MULTIPROCESSOR

Frank Dehne
Andreas Fabri
Mostafa Nassar
Andrew Rau-Chaplin
Rada Valiveti

28 Septembre 1992

Institut National
de Recherche
en Informatique
et en Automatique

2004 route des Lucioles
B.P. 93
06902 Sophia-Antipolis
France

Construction of d-Dimensional Hyperoctrees on a Hypercube Multiprocessor*

La Construction d'un Hyperoctree de Dimension d sur un Hypercube

Frank Dehne [†] **Andreas Fabri** [‡] **Mostafa Nassar** [§]
Andrew Rau-Chaplin [†] **Rada Valiveti** [†]

Programme 4 : Robotique, Image et Vision.

Keywords: Computational Geometry, Data Structures, Geometric Modeling, Parallel Algorithms, Hypercube Multiprocessor.

* *This work was partially supported by the Natural Sciences and Engineering Research Council of Canada and the ESPRIT Basic Research Actions Nr. 3075 (ALCOM) and Nr. 7141 (ALCOM II).*

† *School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6*

‡ *INRIA — B.P.93 — 06902 Sophia-Antipolis cedex, France*

§ *Jodrey School of Computer Science, Acadia University, Wolfville, Canada B0P 1X0*

Abstract

We present a parallel algorithm for the construction of the hyperoctree representing a d-dimensional object from a set of n $(d-1)$-dimensional hyperoctrees, representing adjacent crossections of this object. On a p-processor SIMD hypercube the time complexity of our algorithm is $O(\frac{m}{p}\log p \log n)$, where m is the maximum of input and output size.

Résumé

Nous présentons ici un algorithme parallèle pour la construction de l'hyperoctree représentant un objet de dimension d à partir de l'ensemble d'hyperoctrees de dimension $d-1$ représentant des coupes de cet objet. Sur un hypercube avec p processeurs le temps de calcul est $O(\frac{m}{p}\log p \log n)$, où m est le maximum de la taille de l'entrée etde la sortie.

1 Introduction

In many applications a description of an object is given as a set of cross sections, e.g. in medical diagnosis planar cross sections are obtained via tomography systems. In this paper we consider the problem of computing the 3-dimensional representation of an object given as a set of cross sections. There are two primary versions of this problem which can be considered depending on what is known about the input. If the cross sections are known to be widely spaced or we have no spacing information at all, i.e. the information about the 3-dimensional object consists only of planar contours extracted from the cross sectional images, then we have an interpolation problem, see for example [Bo88]. Alternatively, if the cross sections are known to lie close together, i.e. they can be considered as interpolations of the volume lying between themselves and the following cross section, then we have a merging problem. It is this later version of the problem that is considered in this paper.

The representation of 2 and 3 dimensional objects has been extensively studied ([R80, M88, S89]). One commonly used representation scheme, based on recursive subdivision, is the quadtree and its many variants. For an overview and bibliography on quadtrees and applications we refer to [S84]. Since in this paper we will be interested in d-dimensional representations, where d may be any constant, we will use an octree variant described in [YS83], called hyperoctrees.

Quadtrees and their higher dimensional extension are efficient spatial data structures widely used in image processing, solid modeling and many other application domains. A large number of algorithms for the construction and manipulation of these spatial data structures have been developed. Since these applications are typically data intensive, the application of parallelism to such a fundamental data structure is of both theoretical and practical interest. Recently, researchers have therefore started to consider algorithms for parallel models of computation.

While some papers ([MCI86, ML86]) consider parallel architectures designed (or reconfigured) particularly for quadtree manipulation, others consider general purpose architectures, such as PRAMs ([BRW88]), mesh-connected computers ([HR89]) and hypercubes ([DFR91, IK92]). Many of the construction and manipulation algorithms given in these papers can be easily adapted to work for d-dimensional hyperoctrees. In particular the hypercube construction algorithms for pointer-based [DFR91] and linear [IK92] representations of quadtrees can be adapted to support the construction of hyperoctrees from multidimensional binary images.

In this paper we describe an algorithm for the more complex task of constructing d-dimensional hyperoctrees from sets of $(d-1)$-dimensional hyperoctrees (called *slices*). Given n slices of width one, the basic algorithm consists of $\log n$ phases. In each phase pairs of adjacent slices are joined to form slices of twice the width. In all phases the slices are represented by "hybrid trees", a data structure in between $d-1$ and d-dimensional hyperoctrees (to be defined later). Note that both the input and output hyperoctrees may either be based on pointer or linear representations. The sequential algorithm has a running time of $O(m \log n)$ [YS83], where m is the maximum of input and output size. The time complexity of our parallel algorithm for a SIMD hypercube with p processors is $O(\frac{m \log n}{p} \log p)$. Our parallel algorithm follows the basic merging strategy of the sequential algorithm given in [YS83]. The

main contribution of this paper is to solve the non trivial problem of merging "hybrid trees" in parallel.

The remainder is organized as follows. In the next section we describe our model of computation and some basic operations. In Section 3 we recall the definition of hyperoctrees and give a formal definition of a new data structure called hybrid trees. Section 4 briefly recalls the sequential algorithm and is followed by a section describing a merging algorithm for hybrid trees. Section 6 describes in detail our parallel hypercube algorithm.

2 The Model of Computation

In this section, we present two abstract models of a SIMD hypercube and some basic algorithms for this type of architecture.

A *SIMD hypercube* of dimension d consists of $p = 2^d$ processors which are indexed 0 through $2^d - 1$. Two processors are connected along dimension i, if and only if the binary representation of their indices differ in exactly the i^{th} bit. The processors are synchronized and may be enabled or disabled to execute a common instruction. Each processor has some local memory. Note that we neither assume a constant amount of memory per processor (as is done by exclusively fine-grained algorithms) nor a fixed non-constant amount of memory (as is assumed by exclusively coarse-grained algorithms). Our algorithms are suitable for implementation on either fine-grained or coarse-grained SIMD hypercubes and therefore could be implemented on machines ranging from Intel's iPSC860 [In] (using additional synchronization) to Thinking Machines Corporation's CM2 [St87].

In the first model, arithmetic operations on each processor and communication between processors which are adjacent along a fixed dimension are elementary instructions and take time $O(1)$.

In this paper we will use many *basic vector operations* such as parallel prefix, monotonic routing and bitonic merge [NS81, B68]. The parallel prefix sum of a vector V is the vector W, with $W[k] := \sum_{i=0}^{k} V[i]$, $0 \leq k < p$. Instead of summing we can perform any binary associative operation, e.g. copying. A further generalization is the segmented parallel prefix. The vector is split up in segments and the parallel prefix starts at the beginning of each segment. In this paper we also use an operation called segmented broadcast, that is in each segment all elements are replaced by the first element in the segment.

Monotonic routing refers to the following operation. Given a data vector V, a destination vector D and a Boolean vector selecting some elements of V, the selected data elements $V[i]$ are moved to $V[D[i]]$ under the condition that $D[i] < D[j]$ for all selected elements $0 \leq i < j < p$. That is, the selected data elements remain in the same order.

The bitonic merge algorithm transforms a bitonic sequence (the keys are first increasing, then decreasing) into a sorted sequence. This operation can be used to merge two sorted sequences.

All these vector operations are presented for vectors of length p and they take time $O(\log p)$. They can easily be generalized for vectors of length n, $n \geq p$, and

take then time $O(\frac{n}{p}\log p)$.

The second model under consideration is the *pipelined hypercube*. Arithmetic operations again take time $O(1)$, but processors can communicate with all adjacent processors in time $O(1)$. Thus we can pipeline the above basic vector operations and get $O(\frac{n}{p}+\log p)$, $n \geq p$, as time bounds. The CM2 from Thinking Machines Corporation is an example of such a machine [BLM*91].

3 Hyperoctrees and Hybrid Trees

Hyperoctrees [YS83] are a generalization of the well known quadtree data structure. The d-dimensional hyperoctree, or for short 2^d-tree, is a hierarchical data structure for the representation of a d-dimensional discrete grid of sidelength n, where n is assumed to be a power of 2. The total grid is split in 2^d subgrids of half the sidelength. The subgrids are split recursively and recursion stops when a subgrid of uniform colour is reached. Nodes are either grey, black or white, representing nonuniform, black or white subgrids. The level of a node v is defined recursively. It is $\log_2 n$, if v is the root, and $\text{level}(v) := \text{level}(\text{father}(v)) - 1$ otherwise.

For our algorithm we need a further generalization of 2^d-trees, since we want to represent discrete grids where the sidelength of the side in the d-th dimension varies.

To understand why this generalization is necessary consider the following naive algorithm for combining n 2^{d-1}-trees to form a 2^d-tree.

1. Convert each of the 2^{d-1}-trees that represents a slice into an equivalent 2^d-tree.
2. Perform a union operations on all the 2^d-trees formed in the previous step.

The problem with the naive algorithm is that it can result in a considerable and unnecessary "blow up" in the amount of data that must be processed. Consider for example the situation in which all of the $(d-1)$-dimensional slices are completely black. In the first step of the naive algorithm each of the fully black $(d-1)$-dimensional slices would be broken up in n^{d-1} volume elements of sidelength 1, as the width along dimension d is only 1. Thus we would return to a representation in image space. When in the second step of the naive algorithm the union of all n slices is computed the resulting 2^d-tree is of size $O(1)$. It is exactly this unnecessary increase or "blow up" in data size that any efficient algorithm must seek to avoid.

We need to represent not only d-dimensional cubes but also cuboids with the same sidelength k in dimensions 1 through $d-1$ and sidelength $j \leq k$ in dimension d, where j is the number of already joined slices, and k and j are powers of 2.

Definition 1 A *phase i hybrid tree* T^i is a coloured tree with two kind of nodes. The nodes at levels l, $i \leq l \leq \log n$, form a 2^{d-1}-tree and represent *cuboids* of size $(2^l)^{d-1} \times 2^i$, and nodes at levels 0 through i form a forest of 2^d-trees and represent *cubes* of size $(2^l)^d$.

Notice that phase 0 or $\log n$ hybrid trees can be considered as 2^{d-1}-trees or 2^d-trees, respectively. Further note that nodes at level i are at the same time leaves of the 2^{d-1}-subtree and roots of the 2^d-subtrees. Figure 1 shows a phase $i = 2$ hybrid